Run each script listed below using the provided support files. Comment the code, run the script, take screenshots of the outputs, and respond to the short questions. This document must be submitted along with your .py scripts, screenshots, and any output files.

**NOTE: These scripts will need to be pushed to GitHub. If you have reported issues uploading to GitHub, submit your script & document in zip format & please remind me when submitting to BrightSpace**

**whereisit.py**

**whereisit.jpg & whereisit2.jpg   
Command to run the script (if running from directory where script is located):***python whereisit.py whereisit.jpg*

**A. Comment on what each block in the script is doing and provide a screenshot/upload your script.**

**B. Provide a screenshot of the output on the console.**

**A computer screen with white text

Description automatically generated**

**C. Provide a screenshot of the link opened in google maps.**

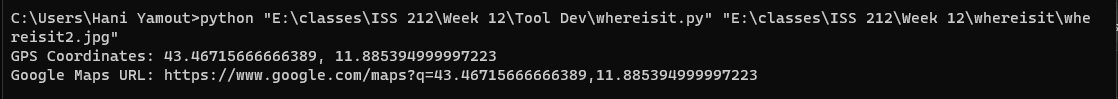
**A screenshot of a computer

Description automatically generated**

**D. What town/city and state was the picture taken in?**

**New Hampshire, Colebrook**

**E. Repeat steps B, C, & D for whereisit2.jpg**



Italy, Via Ricasoli

**F. What mammal is the park named after in the image whereisit.jpg?**

**whatthepdf1.py**

**XMPSpecificationPart3.pdf & whatthepdf.pdf**

” **A. Comment what each block in the script is doing and provide a screenshot/upload your script.**

**B. Provide a screenshot of the output on the console.**

**A screenshot of a computer

Description automatically generated**

**A computer screen with white text

Description automatically generated**

**C. Explain the role of XMP (Extensible Metadata Platform) in PDF documents.**

**In modern PDFs XMP is basically an XML‐based metadata layer that lives inside the file in a special “metadata” stream. It wraps all the info about the document—like title, author, keywords, and any extra custom fields—in a little RDF/XML packet so that any program (even ones that don’t know PDF’s details) can easily find and read it. That packet starts and ends with clear markers (<?xpacket …?>), so you don’t have to parse the whole file. It also keeps the old PDF Info entries in sync with these richer XMP fields, survives incremental saves (so you always get the latest metadata), and makes it easy to add or update custom metadata in a consistent way.**

**bowser.py**C:\Users\{**USERNAME**}\AppData\Local\Google\Chrome\User Data\{**Default or Profile**}\History

**Follow the instruction provided in walkthrough document.  
A. Comment what each block in the script is doing and provide a screenshot/upload your script.**

**B. Provide a screenshot of the text file the history was outputted to.**

**A screenshot of a computer

Description automatically generated**

**(Be sure your screenshot does not contain any sensitive browser history)**

**C. Name and describe, in your words, one of the 4 common tasks that scripting can be used to assist analysts with web browser forensics.**

**Scripting has a crucial role in web browser forensics, especially in browser history analysis, where investigators retrieve and examine a user’s online activity. Browser history is stored in databases, often SQLite, and manually analyzing this data is time-consuming and prone to errors.**

**With scripting, analysts can automate the extraction of URLs, timestamps, and interactions from browser databases. Python scripts can convert complex timestamps (e.g., Chrome’s microseconds format) into readable dates, helping investigators track browsing patterns, suspicious activity, and evidence of cyber threats.**

**Additionally, scripts allow filtering history for specific keywords, such as known malicious sites, and saving structured data in files for further review. This automation is essential in cybersecurity, fraud investigations, and digital crime cases, making forensic work faster, reliable, and scalable. By eliminating manual processes, analysts can focus on interpretation rather than data collection, ensuring accuracy and efficiency.**

**enc-txt.py**

**sup3rs3cr37.txt**

**A. Comment what each block in the script is doing and provide a screenshot/upload your script.**

**B. SAVE YOUR AES KEY as a text file! If you save as a screenshot, you will need to type key out manually.**

**C. Provide a screenshot of the output on the console.**

**A screen shot of a computer

Description automatically generated**

**D. Provide a screenshot of the encrypted file that the script created (It will look like binary txt blob)**

**dc50734be2ae1215026a477f8d4a5cd5ebb8ce8aaaaff8fb9b32bd5cbff8595d**

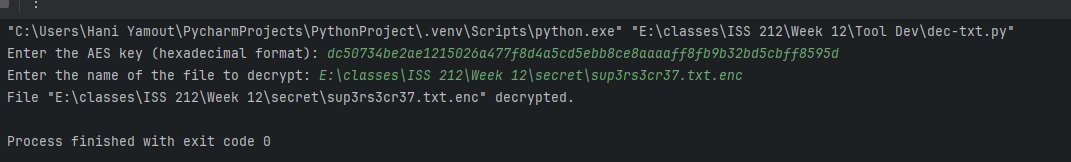
**A screen shot of a computer

Description automatically generated**

**dec-txt.py  
sup3rs3cr37.txt.enc**

**A. Comment what each block in the script is doing and provide a screenshot/upload your script.**

**B. Provide a screenshot of the output on the console.**

****

**C. Provide a screenshot of the message in the decrypted file, sup3rs3cr37.txt\_decrypted.txt.**

**A screen shot of a computer

Description automatically generated**

**stego.py  
drdoes.png**

**A. Comment what each block in the script is doing and provide a screenshot/upload your script.**

**B. Provide a screenshot of the output on the console.**

**C. Until this class, were you familiar with steganography obfuscation techniques? If so, which ones?**

**In an earlier class, we were shown how we could use software tools to find hidden messages that can’t be seen normally if you’re just opening the picture.**

**unstego.py  
{filename you used}.png**

**A. Comment what each block in the script is doing and provide a screenshot/upload your script.**

**B. Provide a screenshot of the output on the console.**

**A computer screen shot of a computer

Description automatically generated**

**C. Name and describe 2 ways threat actors can utilize steganography.**

**Threat actors often use steganography in two main ways to avoid detection and covertly move information. First, they embed malicious payloads—like remote‐access trojans or encrypted commands—inside ordinary image or audio files; because these carriers look and behave like harmless media, they can slip past antivirus scanners and network defenses, then unpack themselves on the target machine. Second, attackers exploit steganography for data exfiltration by hiding sensitive documents or credentials within innocuous-looking pictures or videos; rather than uploading a suspicious file, they can post or email a “normal” image that secretly contains the stolen data, letting it pass through monitoring systems without raising alarms. Both techniques turn everyday multimedia into clandestine communication or smuggling channels, making steganography a powerful tool for evading forensic analysis and persistent surveillance.**

**dca-hw.py**

**dca-log.csv & dca-db.db  
  
A. Comment what each block in the script is doing and provide a screenshot/upload your script.**

**B. Select a user from the list.**

**C. Provide a screenshot of the output on the console.**

**A computer screen with white text

Description automatically generated**

**reg1.py**

**Use the registry hive SOFTWARE provided to you on brightspace.   
Command to run the script (if running from directory where script is located):**

*python reg2.py SOFTWARE*

**A. Comment what each block in the script is doing and provide a screenshot/upload your script.**

**B. Provide a screenshot of the output on the console.**

**A screenshot of a computer program

Description automatically generated**

**reg2.py**

**Use the registry hive SOFTWARE provided to you on brightspace.**

**Command to run the script (if running from directory where script is located):**

*python reg2.py SOFTWARE*

**A computer screen shot of a program

Description automatically generated  
A. Comment what each block in the script is doing and provide a screenshot/upload your script.**

**B. Provide a screenshot of the output on the console.**

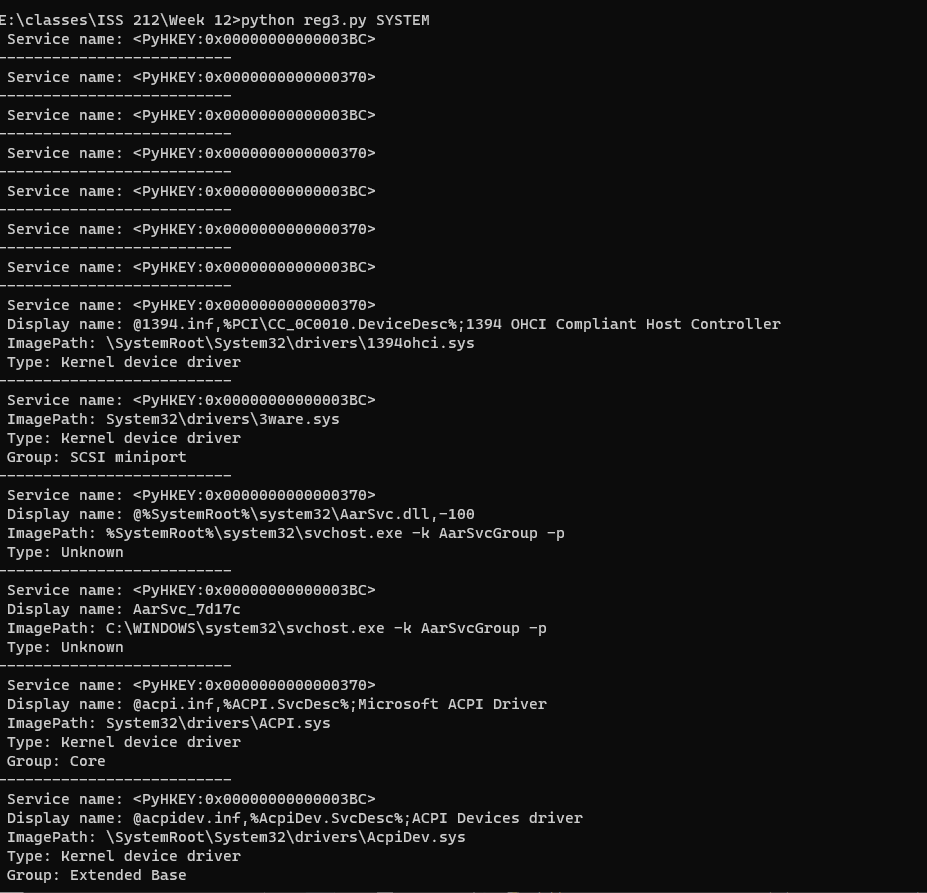
**reg3.py**

**Use the registry hive SYSTEM provided to you on brightspace.   
Command to run the script (if running from directory where script is located):**

*python reg3.py*  SYSTEM

**A. Comment what each block in the script is doing and provide a screenshot/upload your script.**

**B. Provide a screenshot of the output on the console.**

****

**ioclog.py**

**Ioc-bfa.log, ioc-malexec.log, ioc-unauth.log, ioc-scr-yourlastname.txt  
  
Basic logging prompt: ioc-bfa.log**

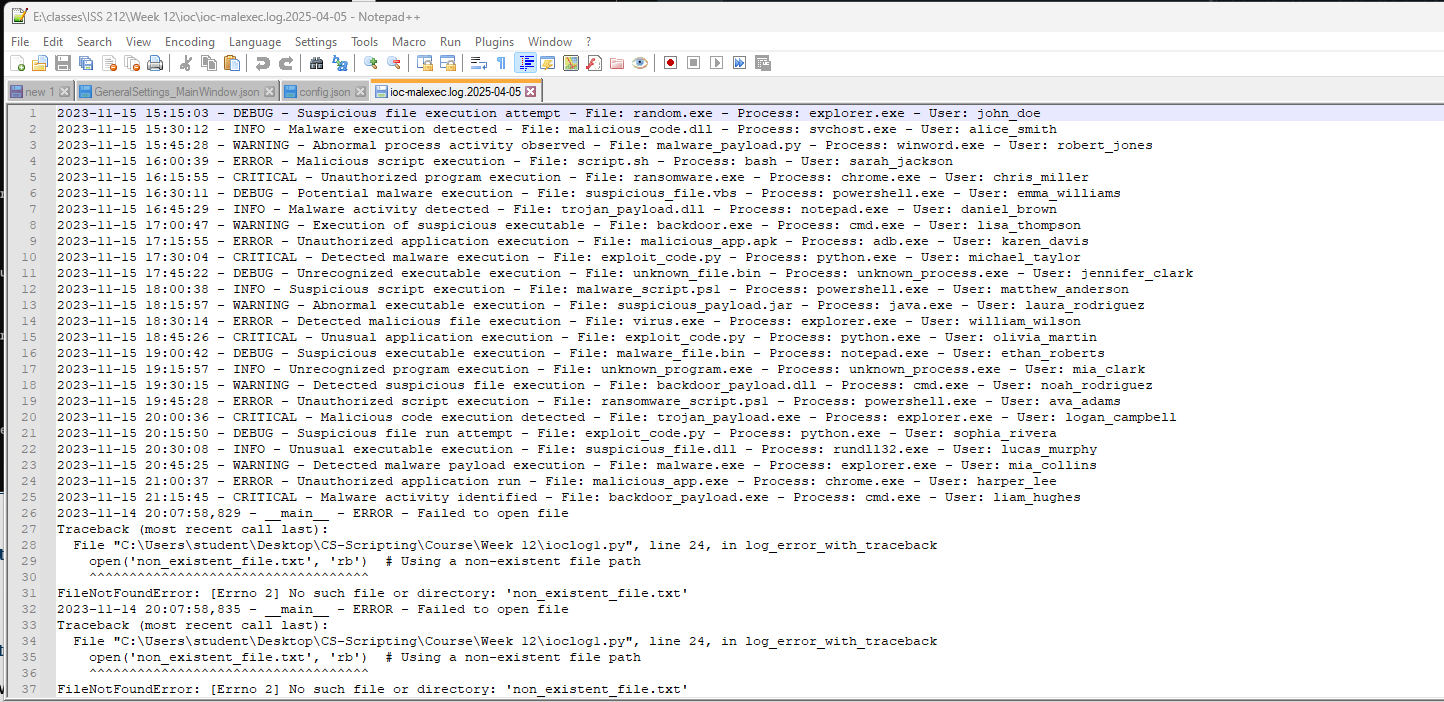
**File rotation logging prompt: ioc-malexec.log**

**Logging errors with traceback prompt: ioc-unauth.log**

**Enter file name to combine all logs prompt: ioc-scr-yourlastname.txt**

**A. Comment what each block in the script is doing and provide a screenshot/upload your script.**

**B. Provide a screenshot of the txt file that was created. (Doesn’t need to be full contents of file)**

****